COMP 472: Artificial Intelligence

Presented to

Dr. Ali Ayub

Image Classification of the CIFAR-10 dataset

By

Khaled Elshokri, 40247895

Concordia University

25th of November 2024

Table of Contents

I. Introduction and Overview:

In this project, our goal is to perform image classification on the CIFAR-10 dataset using different learning algorithms. We will build four AI models using those algorithms, such as, Naïve Bayes, Decision Tree, Multi-Layer Perception, and Convolutional Neural Network. Using basic Python libraries and Pytorch we will be able to classify the 10 object classes in the CIFAR-10 dataset. For each implemented model, we will evaluate the performance and accuracy of the model using a confusion matrix and a table detailing the accuracy, precision, recall, and F1-measure. We will also compare different approaches to each implementation.

II. Data pre-processing:

The data_preparation file prepares the CIFAR-10 dataset to be used in the machine learning models developed in this project. It begins by loading and transforming the CIFAR-10 dataset by resizing, normalization, and conversion to tensors compatible with the ResNet-18 CNN. The dataset is reduced to 500 training and 100 test images per class to make it less computationally expensive, which are then wrapped in data loaders for the batch processing.

A pre-trained ResNet-18 model is used as a feature extractor by removing its final classification layer. Features are extracted for both training and test sets by passing the images through this modified ResNet-18 model, which creates 512-dimensional feature vectors.

Principal Component Analysis (PCA) is applied to reduce the dimensions of the extracted features to 100 dimensions, which improves computational efficiency while retaining the model with different information. The processed features and their corresponding labels are saved as NumPy arrays so they can be used by multiple models without the need of re-extracting the features.

III. Model Architectures and Training:

a. Naïve Bayes:

In the customised Gaussian Naïve Bayes implementation, The means (self.mean), variances (self.var), and priors (self.priors) for each class are explicitly calculated within the fit method. Which is different from the sklearn, there is no implementation for the detection of edge cases for zero variances other than a manual addition of 1e-9 to avoid any possible division by zero. Also, the computation of the logarithmic of priors and likelihoods are made manually using (np.log) to mimic sklearn's implementation. In addition, the implementation of the calculation of the posterior probability for each class is written explicitly and we didn't use the (predict_proba()) method of sklearn. The only hyperparameter that was used in this model is the variance smoothing, where we added a very small constant (1e-9) to make sure we won't run into any divisions by zero.

b. Decision tree:

In the custom decision tree implementation, we randomly select a subset features (max_features) while we are doing the split search to emulate sklearn's implementation but we don't add any optimizations that are used in sklearn's implementation. Sklearn implements the split criteria based on the gini index and other criterias like entropy, but the custom implementation uses only an explicity coded gini index approach for the splitting criteria. Sklean's implementation uses min_sample_leaf and max_sample_leaf as additional stopping criterias, but for our custom implementation we decided to only go with max_depth and min_sample_split to reduce code's complexity.

c. Multi-Layer perception:

The main model for the MLP implementation uses two hidden layers of 512 neurons each, and it includes ReLU activations and batch normalization in the second layer to stabilize the training. It also uses the Stochastic Gradient Descent with momentum (0.9) for better convergence speed. The main model uses Cross Entropy Loss for training, and it uses mini-batch gradient descent by dividing the dataset into batches and updates the model's parameters after each batch. In addition, we shuffle the dataset at the start of each epoch to make sure the model doesn't rely on the order of the training data. The changes to variants will be further discussed in the evaluation section since the variants of this model are the same as the main model but with more or less layers.

d. Convolutional Neural Network:

The main CNN implementation, based on the VGG-11 architecture, introduces some modifications and simplifications to the torchvision.models.vgg11. The custom model explicitly implements batch normalization after every convolutional layer to stabilize and accelerate the training process, which is a feature not included in the standard VGG-11 model. In addition, the classifier head is simplified and adapted for the CIFAR-10 dataset by using a fixed output size of 10 neurons instead of the standard 1000 for ImageNet. The model employs a Cross Entropy Loss for multi-class classification and uses Stochastic Gradient Descent (SGD) with momentum (0.9) to optimize weights. Dropout layers with a probability of 0.5 are included in the fully connected layers to combat overfitting. The dataset uses smaller input sizes (32x32) compared to the standard VGG-11's input size (224x224), and normalization parameters specific to CIFAR-10 are applied. In terms of data handling, the model focuses on simplicity by using resizing and normalization only. The architecture and training methodology remain similar across variants of the CNN, as the main differences (if any) are limited to the addition or removal of layers or small tuning of hyperparameters.

IV. Evaluation of The System:

a. Naïve Bayes:

The following metrics are achieved when testing the custom naïve bayes model:

Table 1: Confusion matrix of the custom Naïve bayes implementation.

Class	airplanes	cars	birds	cats	deer	dogs	frogs	horses	ships	trucks
airplanes	6	6	11	14	16	13	6	12	9	7
cars	8	19	15	1	19	10	9	6	4	9
birds	16	4	9	18	11	7	12	3	9	11
cats	3	16	9	12	8	7	11	16	10	8
deer	6	12	11	2	6	6	8	12	25	12
dogs	7	2	2	23	8	9	18	11	15	5
frogs	9	9	14	11	7	4	11	9	12	14
horses	3	8	8	6	21	6	7	12	16	13
ships	4	13	7	14	8	24	8	11	7	4
trucks	14	23	18	5	11	2	2	14	6	5

*The columns represent the predicted result of the model, the rows are the actual class and the diagonal would represent the perfect prediction.

Table 2: Custom Naïve Bayes main metrics.

Accuracy	Precision	Recall	F1 score
9.6	0.09543207236225529	0.096	0.0957

The following metrics are achieved when testing the Sklearn implementation of the naïve bayes model:

Table 3: Confusion matrix of the Sklearn Naïve bayes implementation.

Class	airplanes	cars	birds	cats	deer	dogs	frogs	horses	ships	trucks
airplanes	6	6	11	14	16	13	6	12	9	7
cars	8	19	15	1	19	10	9	6	4	9
birds	16	4	9	18	11	7	12	3	9	11
cats	3	16	9	12	8	7	11	16	10	8
deer	6	12	11	2	6	6	8	12	25	12
dogs	7	2	2	23	8	9	18	11	15	5
frogs	9	9	14	11	7	4	11	9	12	14
horses	3	8	8	6	21	6	7	12	16	13
ships	4	13	7	14	8	24	8	11	7	4
trucks	14	23	18	5	11	2	2	14	6	5

*The columns represent the predicted result of the model, the rows are the actual class and the diagonal would represent the perfect prediction.

Table 4: Sklearn Naïve Bayes main metrics.

Accuracy	Precision	Recall	F1 score
9.6	0.09543207236225529	0.096	0.0957

When using the Naïve Bayes model for classification, the model fails to identify correlation between features in the images since it takes the approach of conditional independence. So, it most frequently mistakes trucks with cars, or planes with birds, and cats with dogs. Which are pretty similar in feature if each pixel is taken independently from the others. Also, the only class both models correctly identified is the cars and this can be caused by the coincidence that the randomly selected 500 training data had more cars in them so the model had the chance to learn better from more priors and likelihoods.

From the confusion matrix of both implementations of the Naïve bayes model, we observe exact matching of the metrics, since the custom implementation was heavily based on the function implementations of Sklearn, which means adding a small random smoothing factor doesn't have an influence on these metrics.

b. Decision Tree:

The following metrics were achieved using a custom decision tree model with max depth of 50:

Table 5: Confusion matrix of the custom Decision Tree implementation.

Class	airplanes	cars	birds	cats	deer	dogs	frogs	horses	ships	trucks
airplanes	14	14	5	11	8	14	9	14	7	4
cars	10	9	8	12	14	18	6	5	15	3
birds	14	9	9	8	12	10	6	6	12	14
cats	13	11	12	11	14	11	6	8	3	11
deer	9	9	16	10	4	15	6	13	12	6
dogs	9	11	8	11	12	14	6	10	10	9
frogs	12	11	12	7	9	13	7	12	10	7
horses	13	13	8	13	12	13	5	7	6	10
ships	6	19	8	10	11	13	7	5	14	7
trucks	8	21	13	14	7	17	4	8	3	5

*The columns represent the predicted result of the model, the rows are the actual class and the diagonal would represent the perfect prediction.

Table 6: Custom Decision Tree main metrics.

Accuracy	Precision	Recall	F1 score
9.4	0.09449048944869855	0.094	0.09424

The following metrics are achieved using the Sklearn implementation of the Decision Tree model with max depth equal to 50:

Table 7: Confusion matrix of the Sklearn Decision Tree implementation.

Class	airplanes	cars	birds	cats	deer	dogs	frogs	horses	ships	trucks
airplanes	6	14	11	13	6	10	6	14	13	7
cars	11	7	8	12	13	9	10	8	8	14
birds	12	11	11	8	12	10	10	6	11	9
cats	10	10	10	9	13	9	9	11	10	9
deer	12	8	16	10	11	12	5	7	13	6
dogs	15	7	14	8	6	11	4	13	10	12
frogs	9	10	10	12	12	9	7	9	12	10
horses	7	8	6	12	15	8	11	12	13	8
ships	7	7	8	10	9	11	8	18	11	11
trucks	6	9	11	10	11	9	11	14	14	5

*The columns represent the predicted result of the model, the rows are the actual class and the diagonal would represent the perfect prediction.

Table 8: Sklearn Decision Tree main metrics.

Accuracy	Precision	Recall	F1 score
9.0	0.08896379268586962	0.09	0.09424

Usually, Decision Trees perform better than Naïve Bayes on complex datasets, but in our case the decision tree implementation performed similarly to Naïve bayes or even worse. This can be caused by the max depth being 50, which can result in overfitting the training data, which results in a lot of misclassifications. If we modify the maximum depth of the tree to 20 for both implementations, we can observe improvements in the classifications as shown below:

Table 9: Confusion matrix of the custom Decision Tree implementation with Max_Depth = 20.

Class	airplanes	cars	birds	cats	deer	dogs	frogs	horses	ships	trucks
airplanes	14	13	10	15	2	10	10	13	10	3
cars	28	19	8	4	10	8	4	9	5	5
birds	13	13	16	12	11	7	12	9	2	5
cats	16	8	10	9	14	13	8	7	9	6
deer	11	9	18	6	13	10	10	6	7	10
dogs	10	15	8	17	1	6	15	13	4	11
frogs	18	10	11	10	17	7	8	4	4	11
horses	9	11	7	14	13	9	16	7	8	6
ships	17	13	10	11	11	5	8	7	11	7
trucks	15	11	11	18	8	11	3	9	8	6

*The columns represent the predicted result of the model, the rows are the actual class and the diagonal would represent the perfect prediction.

When changing the maximum depth of the tree to 20, it seems to capture more detailed features of each class resulting in better classification metrics as shown below:

Table 10: Custom Decision Tree main metrics with maximum depth set to 20.

Accuracy	Precision	Recall	F1 score
10.9	0.10885142841975126	0.109	0.1089

Similar to the Naïve Bayes implementation, the decision tree model confuses classes that have similar features like cats and dogs, birds and airplanes, etc. An improvement to the decision tree approach could be the use of a Random forests which mitigate overfitting by averaging the predictions of multiple trees trained on different subsets of the data and features.

c. Multi-Layer Perception (MLP):

The following metrics were achieved using the MLP model with ReLU activation and Batch normalisation running for 40 epochs:

Table 11: Confusion matrix of the three-layer MLP implementation.

Class	airplanes	cars	birds	cats	deer	dogs	frogs	horses	ships	trucks
airplanes	7	4	14	19	14	20	6	8	3	5
cars	2	3	11	16	16	22	5	3	7	15
birds	7	0	8	10	21	20	10	11	6	7
cats	2	3	5	10	11	20	28	12	4	5
deer	5	5	8	11	25	22	8	7	6	3
dogs	6	2	11	12	17	18	12	11	4	7
frogs	7	5	3	6	16	17	24	9	9	4
horses	6	3	14	8	17	14	14	9	3	12
ships	7	2	10	18	11	38	4	3	3	4
trucks	8	4	13	12	11	25	8	7	5	7

*The columns represent the predicted result of the model, the rows are the actual class and the diagonal would represent the perfect prediction.

Table 12: Main metrics of the three-layer MLP.

Accuracy	Precision	Recall	F1 score
11.4	0.1100	0.1140	0.11196

The poor performance of the MLP model shown by its confusion matrix and low accuracy (~11%), highlights its unsuitability for image data like CIFAR-10. MLPs treat input data as flattened vectors and loses critical spatial relationships inherent to images. This contrasts with CNNs, which preserve spatial structure using convolutional filters that detect patterns like edges and textures. CNNs also leverage parameter sharing and local receptive fields, allowing them to efficiently learn hierarchical features, but MLPs lack these inductive biases and often struggle with high-dimensional data. Although pre-extracted ResNet-18 features were used as input, the MLP lacked the capacity to fully utilize these rich representations due to its simple architecture and insufficient complexity. The confusion matrix revealed almost random predictions, suggesting the MLP failed to differentiate between classes effectively. Overfitting and the inability to generalize are challenges we found for MLPs, especially when working with complex datasets. To improve the results, strategies like increasing the depth and width of the MLP, adding dropout for regularization, and optimizing hyperparameters could help us achieve better results. Fine-tuning ResNet-18 alongside the MLP may also allow for better adaptation of features. However, even with these changes, CNNs would still likely outperform MLPs due to their ability to learn spatial hierarchies and adapt better to image data.

Increasing the number of layers in the MLP implementation seemed to make the predictions worst. It made the model learn too specifically the data which makes it overfit. The lack of generalization in the model when it overfits makes it unable to correctly classify never seen data thus decreasing the accuracy of the model. When compared to the literature, our MLP seems to be significantly underperforming, which indicates major flaws in the python implementation which makes the model make random predicitons.

d. Convolutional Neural Network:

The following metrics were achieve using the VG11 net trained on the whole CIFAR-10 dataset:

Table 13: Confusion matrix of the VG11 CNN implementation.

Class	airplanes	cars	birds	cats	deer	dogs	frogs	horses	ships	trucks
airplanes	871	10	28	9	22	2	3	7	36	12
cars	9	928	0	4	2	3	1	4	13	36
birds	48	2	738	39	63	36	32	32	8	2
cats	11	5	43	620	54	141	52	57	11	6
deer	7	3	32	28	843	21	23	34	8	1
dogs	11	2	25	73	43	767	20	52	3	4
frogs	5	2	27	33	20	17	878	11	5	2
horses	6	1	9	8	47	21	1	900	2	5
ships	30	12	5	2	5	1	3	9	918	15
trucks	27	52	5	6	2	2	2	5	20	879

*The columns represent the predicted result of the model, the rows are the actual class and the diagonal would represent the perfect prediction.

Table 14: Key metrics of the VG11 CNN implementation.

Accuracy	Precision	Recall	F1 score
83.42	0.8335	0.8342	0.8338

The confusion matrix for the VGG11 CNN implementation shows that the most frequently confused classes are cats and dogs (141 misclassifications) and birds and cats (43 misclassifications). This is likely caused by the shared visual features like fur textures or similar shapes, which can make them harder to distinguish in lower-resolution images like those in CIFAR-10. The model's architecture might not fully capture nuanced differences between these closely related classes. On the other hand, cars and airplanes are well-recognized, probably due to their distinct shapes and strong and easily detectable edges. Their success proves the model's ability to learn high-level spatial features.

Reflecting on depth of the VGG11 CNN, it demonstrated a strong ability to learn detailed features by achieving an accuracy of 83.42%. The model's depth likely helped capture hierarchical feature representations, but there is a risk of overfitting if layers are excessively deep. For the MLPs, increasing layer size often improves performance up to a certain point, beyond which diminishing returns or overfitting occur. In CNNs, kernel size variations impact performance where smaller kernels capture fine-grained details, but larger kernels capture broader patterns. A balance between these factors is important for optimal recognition. The metrics suggest that the VGG11 implementation has a reasonable balance, but refinements of the kernel size and layer depths could further mitigate class confusions.

The following is an implementation of a VG11 CNN using a kernel size of 1, instead of kernel size 3 used in the previous version. A kernel of size 1 doesn't require any padding so we have to set the padding to 0.

Table 15: Confusion matrix of the VG11 CNN implementation with Kernel size 1.

Class	airplanes	cars	birds	cats	deer	dogs	frogs	horses	ships	trucks
airplanes	637	27	68	34	36	12	8	20	101	57
cars	32	657	13	42	5	11	12	18	30	180
birds	70	16	473	87	161	46	79	38	16	14
cats	33	27	63	535	65	121	70	25	30	31
deer	19	4	96	71	602	30	103	53	13	9
dogs	22	22	61	276	62	420	37	50	28	22
frogs	12	7	57	97	91	19	679	25	4	9
horses	44	15	39	92	96	100	30	509	22	53
ships	91	24	20	36	24	23	7	7	703	65
trucks	40	133	9	50	11	11	13	21	64	648

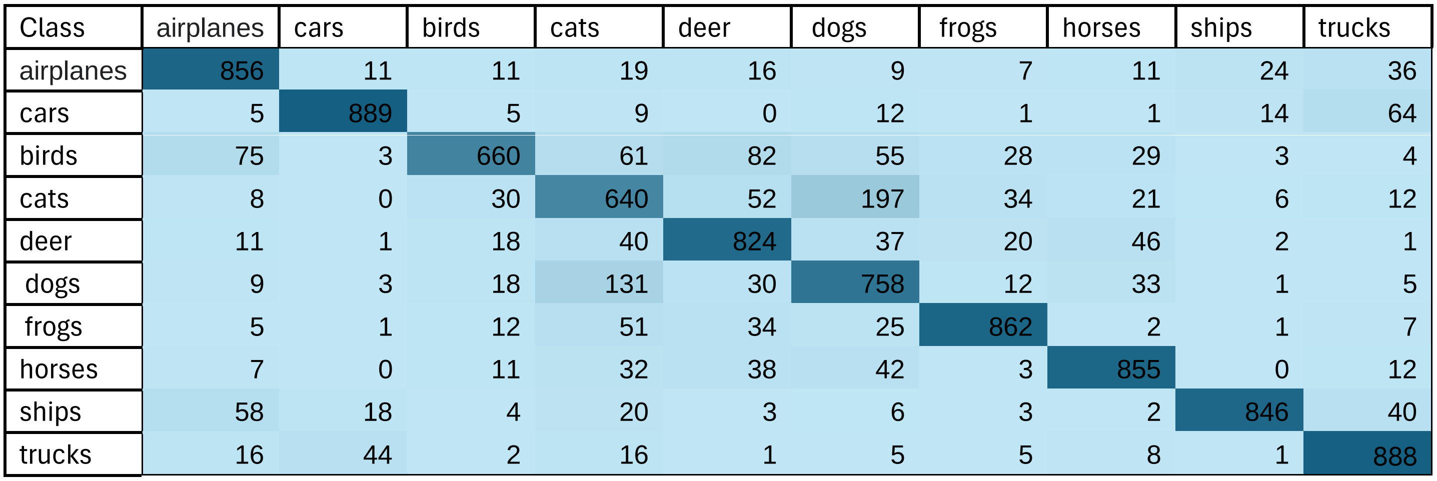
*The columns represent the predicted result of the model, the rows are the actual class and the diagonal would represent the perfect prediction.

Table 16: Key metrics of the VG11 CNN implementation with kernel size 1.

Accuracy	Precision	Recall	F1 score
53.63	0.5935	0.5863	0.589878

A smaller kernel size indicates a decrease in the accuracy of the model. Since a smaller kernel can miss larger features within an image and only focuses on smaller regions within the image. On the other hand, trying with a bigger kernel size like a size 7 with padding of 3, we get the following results:

Table 17: Confusion matrix of the VG11 CNN implementation with Kernel size 7.



Class	airplanes	cars	birds	cats	deer	dogs	frogs	horses	ships	trucks
airplanes	856	11	11	19	16	9	7	11	24	36
cars	5	889	5	9	0	12	1	1	14	64
birds	75	3	660	61	82	55	28	29	3	4
cats	8	0	30	640	52	197	34	21	6	12
deer	11	1	18	40	824	37	20	46	2	1
dogs	9	3	18	131	30	758	12	33	1	5
frogs	5	1	12	51	34	25	862	2	1	7
horses	7	0	11	32	38	42	3	855	0	12
ships	58	18	4	20	3	6	3	2	846	40
trucks	16	44	2	16	1	5	5	8	1	888

*The columns represent the predicted result of the model, the rows are the actual class and the diagonal would represent the perfect prediction.

Table 18: Key metrics of the VG11 CNN implementation with kernel size 1.

Accuracy	Precision	Recall	F1 score
80.78	0.8131	0.8078	0.8104413

From the results shown in table 18, we observe an increase in accuracy compared to the kernel size being 1. Which could be explained by the fact that deeper layers in the network can make the kernel size of 1 more efficient, at the same time, a big kernel size such as 7 can decrease accuracy due to the loss of detail in the images.

V. Conclusion of The System:

With an accuracy of 83.42%, the VGG11 Convolutional Neural Network (CNN) greatly beat the other models in the evaluation of different models for categorising the CIFAR-10 dataset. This performance was much better than that of Naïve Bayes, Decision Trees, and Multi-Layer Perceptrons (MLPs). Due to its incapacity to capture intricate feature interactions and spatial hierarchies, Naïve Bayes and Decision Trees have poor accuracy (~9–10%), with the Decision Tree overfitting at deeper depths. Despite using pre-extracted ResNet-18 features, the MLP's performance was hindered by its restricted complexity, overfitting, and inability to utilise spatial structure, resulting in somewhat improved results (11.4% accuracy). The VGG11 CNN, on the other hand, performed exceptionally well by learning hierarchical features, generalising well to unknown data, and maintaining spatial relationships through convolutional filters. Although it still had trouble with visually identical classes like cats and dogs, its architecture let it to handle the complexity of image data and discriminate between classes with different attributes, such cars and aeroplanes. This demonstrates CNNs' supremacy for image classification tasks, especially for datasets like CIFAR-10 where the extraction of spatial and hierarchical features is crucial.